smsqzine

Issue #1 **June 2015**

SMSQzine table of contents

Published by:
Timothy Swenson
swenson_t@sbcglobal.net
swensont@lanset.com

SMSQzine is published as a service to the Sinclair QL community. Writers are invited to submit articles for publication. Readers are invited to submit article ideas.

Created using Open Source Tools:

- OpenOffice
- Scribus
- Gimp
- SMSQmulator

Why SMSQzine

I've been using the QL for almost 30 years. I bought my first QL in April 1986. Over the years I have contributed articles to a number of newsletters, from local user group newsletters, like TimeLinez, to national newsletters, like Update!, and then international newsletters, like IQLR and QL Today. I even have published and edited my own newsletter, QL Hacker's Journal.

My interest in the QL has ebbed and flowed as other hobbies have taken up my time. At the moment it is more a flow than an ebb and I am at a time where I'm tinkering with and programming on the QL. As I work on projects, I like to write about what I am doing. I saw the need for a new QL newsletter for long form writing, esp. one that is freely available to all.

When I started the QL Hackers' Journal in 1991, I concentrated on the content and ignored the look and style of the newsletter. Since most people were going to get the newsletter via e-mail, keeping it simple was the best. Now with more download bandwidth, the PDF standard, and good open source desktop publishing tools, I plan to make this newsletter more colorful and graphical.

SMSQzine is a free e-zine (electronic magazine). There is no costs to acquire it nor will there be any payment for authors. This whole project will cost only time, for both the reader and writer. There are few QL vendors left, so if any are interested in running an ad, it will be run at no cost.

I am interested in getting feedback from the QL community, both in what I'm doing wrong and what I'm doing right. Ideas for article is invited, along with articles themselves. The topics of articles can be anything that is QL related. I'm hoping that the QL community will find the ezine interesting, entertaining and useful.

Pascal Compiler Review

Computer One Pascal & Prospero ProPascal

I have one program that I've been porting to different compilers on the QL. After porting it to Small-C, Digital C, and QC, I decided to port it to Pascal. There are two freeware Pascal compilers for the QL, Computer One Pascal and Prospero ProPascal.

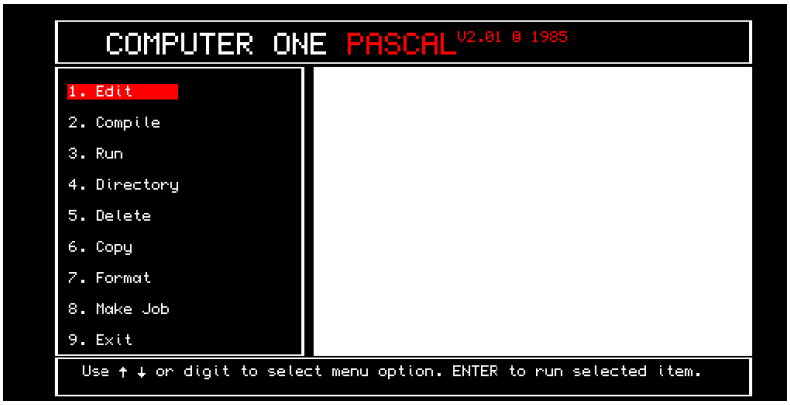
Pascal was published in 1970 by Niklaus Wirth and was the primary language for universities in the late 70's and early 80's. My compiler, data structures, and numerical analysis classes all used Pascal. The release of Turbo Pascal for CP/M and DOS in 1983, made Pascal a very popular language for development.

Both of these compilers are tied to the QL and need an emulator like Qemulator and will not run on an emulator like SMSQmulator. Computer One Pascal seems to rely on the screen position in memory and ProPascal needs to have an ROM file attached to the emulator (the original used a ROM cartridge).

Computer One Pascal

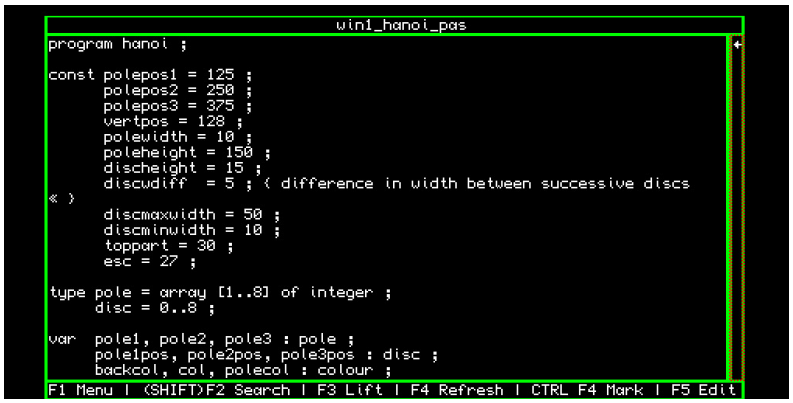
Computer One (C1) of Cambridge produced a number of language utilities for the QL, including as Assembler, Forth, and Pascal.

C1 Pascal is a p-code compiler, meaning that it does not compile to assembly or machine code that is run on the system, but to a secondary language that is then interpreted by a p-code runtime system.



Main menu screen of C1 Pascal

C1 Pascal used a menu-based Integrated Development Environment (IDE). The boot file



C1 Pascal editor screen

brings up the IDE, from which all actions needed to write and compile a Pascal program is provided. The IDE has a built in full-screen editor. If there are any errors in the compilation process, the next time the program is brought into the editor, the errors are shown. The errors are not added to the source code, but only show up via the editor and are not something that can be edited out. When saved, the errors are not included. This was a little disconcerting at first, until I realized what was going on.

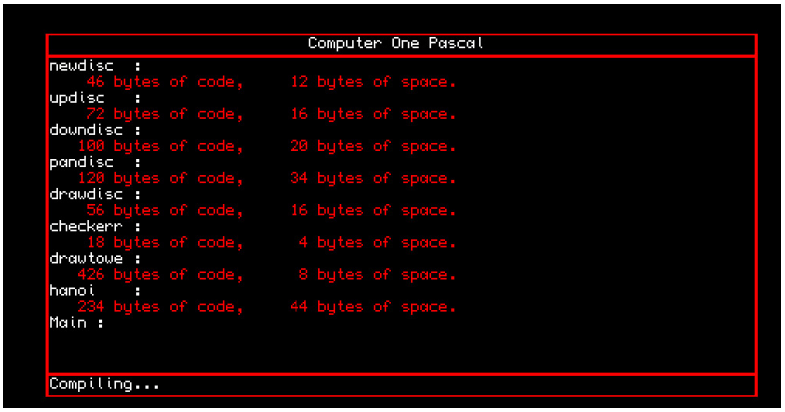
After editing the source code, the next option is to compile to code. If all goes well, the code is compiled into the p-code binary. The binary can be run from the IDE menu for testing. Once the code is complete, a self-running executable can be created so that the program will run on other systems. The other systems will need the Pascal environment toolkit to be loaded to run the executable.

C1 Pascal creating executable

Prospero ProPascal

Prospero was a London based software house that produced programming tools and released ProPascal and ProFortran-77 for the QL.

ProPascal does not have an IDE like C1 Pascal, but is more of a traditional compiler with the compiler and linker called from the command line. There is no included editor, so the user would use their own. With no built in IDE, using MicroEmacs with ProPascal would come close to an IDE, since MicroEmacs can execute QDOS jobs from within the editor and they could be automated with the scripting language that MicroEmacs supports.



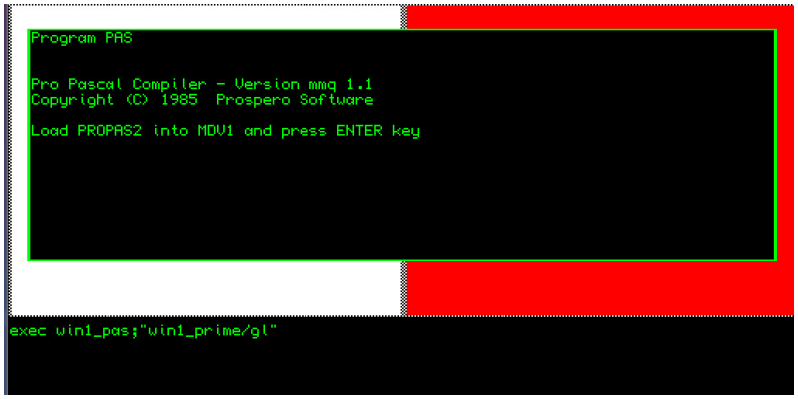
C1 Pascal compiling code

Comparison

Both compilers come with manuals that have enough information to get going. The manuals will not teach one how to program in Pascal, but they provide the details on the version of Pascal supported by the compiler. C1 Pascal manual is 104 pages and ProPascal is 203 pages.

Both compilers support Standard Pascal, but it is a fairly limited language with a limited list of library functions and procedures (about 27 in total). C1 Pascal adds a whole library of 54 QL related functions. These cover QL specific routines to access windows, do QL graphics, PEEK and POKE, etc, plus a few useful routines like rnd, readclock, and upper and lower functions. The

commands are fairly complete and porting SuperBasic programs to Pascal should be easy.



ProPascal compile screen

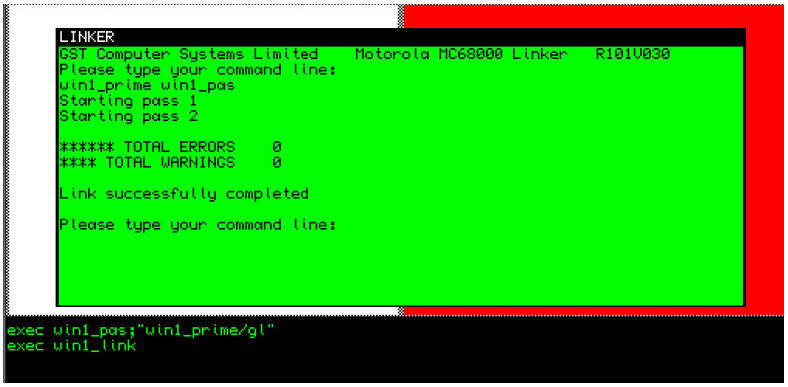
ProPascal covers the same Standard Pascal functions and procedures, the same QL graphics routines and then adds more, like execprog, where one Pascal program can execute another program. In general the ProPascal supported library is slightly larger than C1 Pascal.

When using library routines, C1 Pascal compiles the included routines with no issues. For ProPascal, there is a graphics library definition file that must be included in a Pascal program for the compiler to recognize the routines. There are also a number of procedures that must manually be listed in the program as EXTERNAL, before the compiler will recognize the procedure. Why there was not a single include file with all of the library procedures, I'm not sure. It would seem the right thing to do to make it easier on the programmer.

The biggest difference between the two compilers is how windows are handled. In C1 Pascal, a window can be opened as a console device, meaning that it is for both input and output. In ProPascal, any window is a screen (scr_) device, meaning that it is only for output. ProPascal opens a default window, that is a console, to get user input. If the programmer wants to get input later on after a window is opened, they are out of luck.

When compiling C1 Pascal stops at each error and gives you the opportunity to stop compilation or to continue. If no input is received within about 15 seconds, the compilation process continues.

When compiling with ProPascal, any errors are reported on the screen, but there is no pause, so they scroll by. There is an option to send a log of the compilation process to a file, so that it can be viewed later. ProPascal also has a 2 stage compilation. When on microdrive, each part of the process was on a separate drive, and the program paused for the microdrives to be switched out. When using a disk system, the pause still happens, and all that is needed is to hit the Enter key.

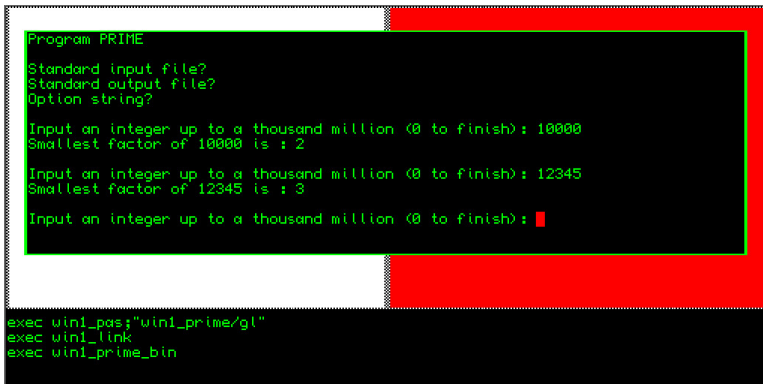


ProPascal linker screen

Conclusion

For someone new to Pascal, I would recommend C1 Pascal. The IDE menu system makes it easy to use the compiler. The addition of the error messages to the source code takes a little getting used to, but it is helpful. The included library is more limited than ProPascal, but I found it enough to get what I needed done.

ProPascal is a more versatile compile, with command line execution that allows for automation by scripting, but it is a more



Default screen for ProPascal

difficult package to use. To review errors you have to look at the log file. The key issue for me is the output-only windows. If writing non-graphical program, such as filters, then this compiler might be the better option.

I did find that it was fairly easy to take a Pascal program written for one compiler and get it working on the other compiler. This means that is one compiler is not suiting your needs, it is fairly easy to move to the other and keep working.

I did have one function that I needed that neither compiler had. I needed to take string input from the user and convert it into a decimal number. The function atoi (ascii to integer) is listed on this page.

```
function atoi (strg : string; len: integer): integer;
   var  a, x, z, count : integer;
        y : real;

   begin
      count := 0;
      x := len;
      while ( x > 0) do begin
        if (len = x) then z := 1
         else begin
            a := len-x;
            y := exp(a*ln(10)); {* 10^x *}
            z := trunc(y);
         end; {* end else *}
         count := count + ((ord(strg[x]) - ord('0'))*z);
         x := x - 1;
      end; {* end while *}
      atoi := count;
   end; {* end function *}
```

Datamining with Archive

Of the four applications that came with the QL, Archive was probably the most powerful and the most under utilized. I started playing with databases in 1983 with dBase II, at the time, the most popular database for DOS and CP/M. dBase is a lot like Archive, in that you have a sparse looking front-end with a command line. In fact, dBase was even sparser with just a dot prompt and no menu across the top. The power of dBase was its programming ability.

Data mining is trolling through large databases looking for trends and grouping information. When looking at Archive for data mining, there is the issue of how many records the it can handle. Luckily, Bill Cable did research on the limits of Archive with sorting / order on none, one, or more fields.

Sort / Ordering	Max Records
No ordering	64,000 records
One sort field	4,670 records
Two sort fields	2,974 records
Three sort fields	2,174 records
Four sort fields	1,710 records

If these limits are exceeded, Bill reports that Archive will lock up.

For tinkering around, I decided to use a data set that I already had, the 1930 Census of Alvarado, part of the town I currently live in. I already had the census in a .csv format, which is what Archive uses for an Import format. The census has 1,889 records,

The first step was the make sure that the data was ready for Import. In the first review the data looked fine. As I imported the data, I ran into bad parts of the data file. When importing to Archive, if there is an issue, Archive will generate an error. It will stop the import at the record that is failing. It will create the database on disk.

My process for resoving the issue was this:

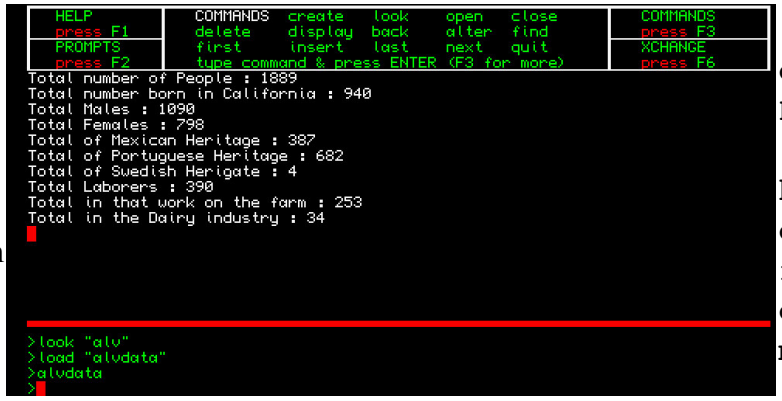
```
     > import "nfa1_alv_csv" as
"win1_alv_dbf"
          - get an error
     > display
          - see what record had an issue.
          - Edit the record using a text editor to
fix the issue
          - CTRL-C back to the Windows 0.
     delete win1_alv_dbf
          - Start the process over again.
```

The problems that I had was that some of the numeric fields had the data stored in text. The date 1875 would show up as "1875". I edited out the quotes and all was fine.

Once I had the data into Archive as a database, I could then run the "select" command to run a query on the database. At the top level the query might be "How many people in the census are female?". There is a field called sex$, where the value M is for Male and F for female. The query was like this:

```
> look "alv"
> select sex$ = "F"
> print count()
```

The count() function lists the total number of records that the select returns. In SQL terms, the select command creates a View of the database. Once it is run, all actions on the database is only done of those records that were picked by the select command. To return to looking at the whole database, the reset command is run.



Since Archive has a programming language, it is possible to create a report in the form of a procedure that automates the queries, this way they don't have to be typed in each time.

With the data that the census has, I can find out a fair number of things. I can find out how many people were born in or had parents born in a certain country. I can find out how many where born in California. I can find out how many people were working in a specific industry. It would take more coding but it would be possible to do an average age of the residents of Alvarado. I could group the people by age to see how many there are in each age bracket.

```
proc alvdata
   print "Total number of People :
";
   select lname$ <> ""
   print count()
   reset
   print "Total number born in
California : ";
   select birth$ = "California"
   print count()
   reset
   print "Total Males : ";
   select sex$ = "M"
   print count()
   reset
   print "Total Females : ";
   select sex$ = "F"
   print count()
   reset
   print "Total
of Mexican
Heritage : ";
   select
birth$="Mexico"
or
fbirth$="Mexico"
or
mbirth$="Mexico"
   print count()
   reset
   print "Total of Portuguese
Heritage : ";
   select birth$="Portugal" or
fbirth$="Portugal" or
mbirth$="Portugal"
   print count()
   reset
   print "Total of Swedish
Herigate : ";
   select birth$="Sweden" or
fbirth$="Sweden" or
mbirth$="Sweden"
   print count()
   reset
   print "Total Laborers : ";
   select trade$="Laborer"
   print count()
   reset
   print "Total in that work on
the farm : ";
```

```
   select industry$="Farm"
   print count()
   reset
   print "Total in the Dairy
industry : ";
   select industry$="Dairy"
   print count()
   reset
endproc
```

Printing with Emulators with Linux

On my laptop, I have two emulators that I use; SMSQmulator and Qemulator. I mostly use SMSQmulator, but there are some programs that require an emulator that better emulates the QL hardware, so I use Qemulator for them.

Occasionally, I have to print from a QL program. Since I'm not using a black box QL with a printer attached, printing is not a trivial task. So I sat down and figured out a way to do that.

Most of the time I'm printing from Xchange, which does allow for printing to a file. Other QL software should have an option for this. I'm not trying to print anything fancy, like graphics, but mostly text from Quill, Abacus or Archive. The default extension for a print file in Xchange is "_lis". With the output to a file, the next step is how to get that file to the underlying operating system.

```
#!/bin/csh

while ( 1 )
   set files = `ls -1 *_lis`

   if ($status == 0) then
      foreach filee ($files)
         lpr $filee
         rm $filee
         sleep 5
      end
   endif
   sleep 30
end
```

Both SMSQmulator and Qemulator can link a device to a directory on the laptop. For SMSQmulator, there is the NFA device. In SMSQmulator, click on Config in the toolbar. Then select "Set Dirs for NFA Drives" and enter a directory name. This will be the device NFA1_.

In Qemulator, click on one of the empty microdrives below the main screen and select "Attach Directory". Then browse to the right directory and select OK. If you clicked on the 3rd microdrive, then the directory will be accessible as mdv3_, win3_, or flp3_.

To get a print file to the local file system, just have Xchange send the print file to it. For SMSQmulator, when Xchange asks for the name of the file, enter "nfa1_test_lis". Xchange will open the NFA1_ device and sent the output to the file test_prn. Checking the NFA directory shows a file called "test_lis". For Qemulator, it would be "win3_test_lis".

The next step is to get this to a printer. Since I'm running Linux, this is very simple. From the shell, run the command:

```
% lpr test_lis
```

the lpr command (which stands for line printer), sends the file to the default printer as defined by CUPS, the Linux printer configuration system. My printer is a network printer in the garage, but CUPS knows how to print to it. The printer speaks PCL, which is backwards compatible with the older Epson escape printer command set. Withing a minute, the printer has printed out the test_lis file.

To do anything beyond the basic, I would set up Xchange to use commands that my specific printer knows (such as underlining, bold face, etc).

To print to a PostScript printer, Ghostscript can be used to convert plain text to PostScript and then send have that to the printer. It should be possible to run a ghostscript command and have the output sent to lpr to be sent to the printer.

One problem with this process is the extra step of leaving the emulator and entering the command to send the file to the printer. It is possible to create a shell script that can to that automatically. The script would have to be started before trying to print, or it could be added to a start up script so that it is executed at boot time.

The script will run forever until it is stopped. It is constantly checking the current directory for a file with an _lis extension. Once it finds one, it send it to the print and then deletes the file. To keep it from taking up too much cpu cycles, the script pauses for 30 seconds between each cycle of checking the directory.

Alternate Configurations with SMSQmulator

A number of early QL programs were designed to boot from microdrive or floppy and sort of take over the QL. They would install toolkits that might not be compatible with other toolkits. Set up the screen to what they need and make it difficult to run other programs.

For programs like these, I have used Qemualtor and used different configuration files (_QCF). When I start Qemulator, I can open one of a number of configuration files, which sets up the emulator with the right QXL.win files and other settings.

With SMSQmulator, there is only one configuration file, SMSQmulator.ini in the user home directory. If I need to alter the configuration of SMSQmulator, I have to go into the config menu and make the changes. It does not make it easy to switch between different configurations.

If I want to occasionally test SMSQmulator with a large screen, I have to manually change it and then manually change it back. I was thinking about how to use something similar to the QCF file for Qemulator.

I then realized that I can keep different copies of the SMSQmulator,ini config file. I could name one big.ini and one normal.ini. Depending on what configuration I wanted to use, I could just copy one of the .ini files over the SMSQmulator.ini file. So, the SMSQmulator.ini file would not be permanent, but the big.ini and normal.ini are the real configuration files. A simple copy of one file to SMSQmulator.ini and I have the change in configuration. When done, copy the other one to SMSQmulator.ini and I'm back to my other configuration.

The configuration can have more then just the display size. It can have a whole different set of QXL.win files, which would have different boot files.

The copy process is fairly simple to do. The next step is to create a couple of shell scripts that would copy a specific .ini file and then execute SMSQmulator. I created a big.sh that copied a big screen configuration .ini file, and normal.sh that copied the normal size configuration .ini file.

To make it easier, I created a desktop shortcut that executed either one of the scripts. Now, I just have to click on an icon, and the configuration of SMSQmulator will be copied and the emulator executed.

big.sh:

```
#!/bin/sh
cp big.ini SMSQmulator.ini
java -jar
/home/swensont/ql/smsqmulator/SMSQ
mulator.jar
```

normal.sh:

```
#!/bin/sh
cp orig.ini SMSQmulator.ini
java -jar
/home/swensont/ql/smsqmulator/SMSQ
mulator.jar
```